5. Transformations of Objects

- to develop techniques for working with affine transformations.

5.1 Affine Transformations

- Affine space: linear space with a coordinate frame

- Affine transformation: a linear function (transformation) from an affine space to another affine space

- a function \( f(\ ) \) from an affine space \( A \) to an affine space \( B \) is said to be a linear function if and only if, for any scalars \( \alpha \) and \( \beta \), and any entities \( E \) and \( F \) (vectors or points) of the domain space \( A \),

\[
f(\alpha E + \beta F) = \alpha f(E) + \beta f(F)
\]

and \( \alpha E + \beta F \), \( \alpha f(E) \) and \( \beta f(F) \) are well defined.
- An affine transformation maps a point to a point and a vector to a vector

- an affine transformation from an affine space $A$ to an affine space $B$ can always be written as

$$F = M \cdot E$$

where $M$ is a square matrix, and $E$ and $F$ are homogeneous representations of two points or two vectors

Why?

Frame of $A$: $\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix}$ \quad \rightarrow \quad Frame of $B$: $\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$
Let

\[ f(\mathbf{u}_1) = m_{11} \mathbf{v}_1 + m_{21} \mathbf{v}_2 + m_{31} \mathbf{v}_3 \]
\[ f(\mathbf{u}_2) = m_{12} \mathbf{v}_1 + m_{22} \mathbf{v}_2 + m_{32} \mathbf{v}_3 \]
\[ f(\mathbf{u}_3) = m_{13} \mathbf{v}_1 + m_{23} \mathbf{v}_2 + m_{33} \mathbf{v}_3 \]
\[ f(Q_0) = m_{14} \mathbf{v}_1 + m_{24} \mathbf{v}_2 + m_{34} \mathbf{v}_3 + P_0 \]

i.e.,

\[
\begin{bmatrix}
    f(\mathbf{u}_1) \\
    f(\mathbf{u}_2) \\
    f(\mathbf{u}_3) \\
    f(Q_0)
\end{bmatrix} = M^t
\begin{bmatrix}
    \mathbf{v}_1 \\
    \mathbf{v}_2 \\
    \mathbf{v}_3 \\
    P_0
\end{bmatrix}
\]

where

\[
M^t = \begin{bmatrix}
    m_{11} & m_{21} & m_{31} & 0 \\
    m_{12} & m_{22} & m_{32} & 0 \\
    m_{13} & m_{23} & m_{33} & 0 \\
    m_{14} & m_{24} & m_{34} & 1
\end{bmatrix}
\]
If \( P = E^t \) is a point in the affine space \( A \) where \( E \) is the homogeneous representation of \( P \)

\[
E^t = \begin{bmatrix} P_x & P_y & P_z & 1 \end{bmatrix}
\]

and \( Q = f(P) = F^t \) is the image of \( P \) in the affine space \( B \) where \( F \) is the homogeneous representation of \( Q \)

\[
F^t = \begin{bmatrix} Q_x & Q_y & Q_z & 1 \end{bmatrix}
\]
then we have

\[
f (P) = f (E^t \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix}) = f (P_x u_1 + P_y u_2 + P_z u_3 + Q_0)
\]

\[
= P_x f (u_1) + P_y f (u_2) + P_z f (u_3) + f (Q_0)
\]

\[
= E^t \begin{bmatrix} f (u_1) \\ f (u_2) \\ f (u_3) \\ f (Q_0) \end{bmatrix} = E^t M^t = F^t \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}
\]

Hence, \( F = M \cdot E \)
• Affine transformations of vectors have $9$ degrees of freedom

\[
\begin{bmatrix}
    v_x \\
v_y \\
v_z \\
0
\end{bmatrix} =
\begin{bmatrix}
    m_{11} & m_{12} & m_{13} & m_{14} \\
m_{21} & m_{22} & m_{23} & m_{24} \\
m_{31} & m_{32} & m_{33} & m_{34} \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
u_x \\
u_y \\
u_z \\
0
\end{bmatrix}
\]

• Affine transformations of points have $12$ degrees of freedom

\[
\begin{bmatrix}
    Q_x \\
Q_y \\
Q_z \\
1
\end{bmatrix} =
\begin{bmatrix}
    m_{11} & m_{12} & m_{13} & m_{14} \\
m_{21} & m_{22} & m_{23} & m_{24} \\
m_{31} & m_{32} & m_{33} & m_{34} \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
P_x \\
P_y \\
P_z \\
1
\end{bmatrix}
\]
5.2 2D Examples of Affine Transformations

- Rotation, translation, scaling and shearing are all Affine transformations

(Rotation and Translation are called rigid-body motion)

**Translation:** \( Q = P + \Delta, \) \( \Delta: \) displacement

\[
\begin{bmatrix}
Q_x \\
Q_y \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & \Delta_x \\
0 & 1 & \Delta_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
P_x \\
P_y \\
1
\end{bmatrix}
\]
Rotation:

\[
\begin{bmatrix}
Q_x \\
Q_y
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
P_x \\
P_y
\end{bmatrix}
\]
Scaling:

\[
\begin{bmatrix}
Q_x \\
Q_y
\end{bmatrix} =
\begin{bmatrix}
S_x & 0 \\
o & S_y
\end{bmatrix}
\begin{bmatrix}
P_x \\
P_y
\end{bmatrix}
\]

\(S_x, S_y\): scaling factors

(get **reflection** if negative scaling factors are used)
Shearing (in $x$ direction):

\[
\begin{bmatrix}
Q_x \\
Q_y \\
1
\end{bmatrix} = \begin{bmatrix}
1 & h \\
0 & 1
\end{bmatrix} \begin{bmatrix}
P_x \\
P_y \\
1
\end{bmatrix}
\]

$h = \cot \theta$: shearing fraction in $x$ direction
Advantage:

Using homogeneous coordinates, all affine transformations can be put in matrix form. Therefore, a sequence of consecutive transformations can be carried out with just one matrix-vector multiplication (by accumulating the corresponding matrices into a single matrix).

Remarks:

- Rotation about an arbitrary pivot point is different from rotation about the origin
- Scaling about an arbitrary pivot point is different from scaling about the origin
- Shearing about an arbitrary pivot point is different from shearing about the origin
**Example:** Rotation About Pivot Point $V$:

1. Translate $V$ to origin
2. Perform rotation at origin
3. Translate back to $V$

**Matrix:**

$$
\begin{bmatrix}
1 & 0 & V_x \\
0 & 1 & V_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -V_x \\
0 & 1 & -V_y \\
0 & 0 & 1
\end{bmatrix}
$$
Example: Scaling About Pivot Point $V$:

Scaling about $V$

1. Translate $V$ to origin

2. Perform scaling at origin

3. Translate back to $V$

Matrix:

\[
\begin{bmatrix}
1 & 0 & V_x \\
0 & 1 & V_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -V_x \\
0 & 1 & -V_y \\
0 & 0 & 1
\end{bmatrix}
\]
**Example:** Shearing About Pivot Point $V$:

1. Translate $V$ to origin
2. Perform shearing at origin
3. Translate back to $V$

**Matrix:**

\[
\begin{bmatrix}
1 & 0 & V_x \\
0 & 1 & V_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & h & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -V_x \\
0 & 1 & -V_y \\
0 & 0 & 1
\end{bmatrix}
\]
5.3 Properties of Affine Transformations

- Affine transformations preserve lines and planes

\[
L(t) = (1-t)A + t B
\]

\[
Q(t) = (1-t)T(A) + t T(B)
\]

- Affine transformations preserve parallelism of lines and planes

\[
L_1(t) = A_1 + b t
\]

\[
L_2(t) = A_2 + b t
\]

\[
Q_1(t) = T(A_1) + T(b) t
\]

\[
Q_2(t) = T(A_2) + T(b) t
\]
The matrix columns of an affine transformation reveal the transformed coordinate frame

\[ M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 \end{bmatrix} \]

\[ \mathbf{m}_1 = M \mathbf{i}, \quad \mathbf{m}_2 = M \mathbf{j}, \quad \mathbf{m}_3 = M \mathbf{Q} \]

Affine transformations preserve relative ratios
Effect of transformations on the areas of figures

\[
\frac{\text{area after transformation}}{\text{area before transformation}} = |\det M|
\]

Every affine transformation is composed of elementary operations

\[
\begin{bmatrix}
  a & c & e \\
  b & d & f \\
  0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & e \\
  0 & 1 & f \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  a & c & 0 \\
  b & d & 0 \\
  0 & 0 & 1
\end{bmatrix} ;
\]

\[
\begin{bmatrix}
  a & c \\
  b & d
\end{bmatrix} = \begin{bmatrix}
  1 & 0 \\
  \frac{ac+bd}{R^2} & 1
\end{bmatrix} \begin{bmatrix}
  R & 0 & a/R \\
  0 & \frac{ad-bc}{R} & b/R
\end{bmatrix}
\]

where

\[
R = (a^2 + b^2)^{1/2}.
\]
5.4 3D Affine Transformations

Translation: \( Q = M \ P \)

\[
M = \begin{bmatrix}
1 & 0 & 0 & \Delta x \\
0 & 1 & 0 & \Delta y \\
0 & 0 & 1 & \Delta z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Scaling: \( Q = M \ P \)

\[
M = \begin{bmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Shearing: \[ Q = M \ P \] (yz shear)

where

\[
M = \begin{bmatrix}
1 & \cot \theta & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Rotation:

**z-roll:** (rotation about \( z \)-axis)

\[
R_z(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**x-roll:** (rotation about \( x \)-axis)

\[
R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**y-roll:** (rotation about \( y \)-axis)

\[
R_y(\theta) = \begin{bmatrix}
\cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Why is the structure of a **y-roll** different from that of an **x-roll** or a **y-roll**?

Consider the following figure:

![Diagram showing projections of x-roll, z-roll, and y-roll](image)

and look at projections of **x-roll**, **z-roll** and **y-roll**:
Rotations about an arbitrary axis:

Given a unit vector $\mathbf{u}$ and a point $P$, how would you find the matrix that represents the rotation of $P$ about the axis defined by $\mathbf{u}$ for $\beta$ degree?

Method 1:

1. Rotate $\mathbf{u}$ about $y$-axis for $\theta$ degree and then rotate about $z$-axis for $-\phi$ degree to align $\mathbf{u}$ with $x$-axis
2. Perform rotation of $P$ about $x$ for $\beta$ degree
3. Reverse step 1 to restore $\mathbf{u}$ to its original direction
Method 2:

view it as a 2D rotation and use vector tools

1. Let \( \mathbf{p} \) be position vector of \( P \) and \( \mathbf{h} \) the position vector of \( O_1 \) where \( O_1 \) is the projection of \( P \) on \( \mathbf{u} \)

\[
\mathbf{h} = (\mathbf{p} \cdot \mathbf{u}) \mathbf{u}
\]

2. Define a 2D coordinate frame \((\mathbf{a}, \mathbf{b}, O_1)\) for the plane determined by \( P, Q \) and \( O_1 \)

\[
\mathbf{a} = \mathbf{p} - \mathbf{h}, \quad \mathbf{b} = \mathbf{u} \times \mathbf{a}
\]
3. Then $Q$ is equal to

$$Q = h + (\cos \beta) \ a + (\sin \beta) \ b$$

$$= h + (\cos \beta) \ (p - h) + (\sin \beta) \ (u \times p)$$

$$= (\cos \beta) \ p + (1 - \cos \beta) \ h + (\sin \beta) \ (u \times p)$$

$$= (\cos \beta) \ p - (1 - \cos \beta)(p \cdot u) \ u + (\sin \beta) \ (u \times p)$$

$$= I(\cos \beta) \ P + (1 - \cos \beta) (u' u) \ P + (\sin \beta) \ Cross (u) \ P$$

$$= \cos \beta \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ P$$

$$+ (1 - \cos \beta) \begin{bmatrix} u_x^2 & u_x u_y & u_x u_z \\ u_x u_y & u_y^2 & u_y u_z \\ u_x u_z & u_y u_z & u_z^2 \end{bmatrix} \ P$$
\[ \sin \beta \begin{bmatrix} 1 & -u_y & u_z \\ -u_z & 0 & u_x \\ u_y & -u_x & 0 \end{bmatrix} P \]

Hence, the homogeneous matrix representation of the rotation is:

\[
R_u(\beta) = \begin{bmatrix}
  c + (1-c)u_x^2 & (1-c)u_xu_y - s u_z & (1-c)u_xu_z + s u_y & 0 \\
  (1-c)u_xu_y + s u_z & c + (1-c)u_y^2 & (1-c)u_zu_y - s u_x & 0 \\
  (1-c)u_xu_z - s u_y & (1-c)u_yu_z + s u_x & c + (1-c)u_z^2 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

where \( c = \cos \beta \) and \( s = \sin \beta \).
Interesting Application:

Given a rotation matrix, can you determine the axis and angle of the rotation?

\[
R_u(\beta) = \begin{bmatrix}
m_{11} & m_{12} & m_{13} & 0 \\
m_{21} & m_{22} & m_{23} & 0 \\
m_{31} & m_{32} & m_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

The trace of \( R_u(\beta) \) equals \( 1 + 2 \cos \beta \). Hence,

\[
\cos \beta = \frac{(m_{11} + m_{22} + m_{33} - 1)}{2}
\]

\( (m_{32} - m_{23}) \) equals \( 2 \sin(\beta) \, u_x \). Hence,

\[
u_x = \frac{(m_{32} - m_{23})}{2\sin\beta}
\]

Similarly,

\[
u_y = \frac{(m_{13} - m_{31})}{2\sin\beta}, \quad u_z = \frac{(m_{21} - m_{12})}{2\sin\beta}
\]
Changing Coordinate Systems

**Theorem:** Suppose coordinate system #2 is formed from coordinate system #1 by affine transformation $M$. Suppose $[P_x, P_y, 1]$ are the coordinates of a point $P$ expressed in system #2. Then the coordinates of $P$ expressed in system #1 are $MP$.

**Proof.** Let the coordinate frames of systems #1 and #2 be $(i', j', Q')^t$ and $(i, j, Q)^t$, respectively. Then

$$[P_x, P_y, 1] \begin{bmatrix} i' \\ j' \\ Q' \end{bmatrix} = P_x i' + P_y j' + Q'$$

$$= P_x (M i) + P_y (M j) + (M Q)$$

$$= M (P_x i + P_y j + Q)$$

$$= (M \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix})^t \begin{bmatrix} i \\ j \\ Q \end{bmatrix}$$
Application: Successive coordinate frame changes

Suppose coordinate system #1 is transformed to coordinate system #2 by affine transformation $M_1$ and coordinate system #2 is transformed to coordinate system #3 by affine transformation $M_2$. If $(e, f, 1)$ is the representation of a point $P$ with respect to system #3, then what is the representation of $P$ with respect to system #1?

\[
\begin{pmatrix}
  a \\
  b \\
  1
\end{pmatrix} = M_1 \cdot M_2
\begin{pmatrix}
  e \\
  f \\
  1
\end{pmatrix}
\]

In REVERSED ORDER. WHY?
IMPORTANT:

To transform points, current matrix is premultiplied by each new transformation

\[ M = M_3 \times M_2 \times M_1 \]

To transform coordinate systems, current matrix is postmultiplied by each new transformation

\[ M = M_1 \times M_2 \times M_3 \]

(This is how OpenGL operates.)
Why?

Every graphics item (point, line, polygon, ...) has to go through the following pipeline:

\[
M = (M_w \ M_p \ M_v) \ M_m
\]

\[
= M_1 \ M_m
\]

- \(M_m\): modeling transformation (modelview matrix)
- \(M_v\): viewing transformation
- \(M_p\): projection
- \(M_w\): window-to-viewport transformation
$M_m$ is the accumulation of the transformations performed on the graphics item by the user in the modeling space.

If object A is transformed by a x-roll $R_x$, a translation $T$ and a scaling $S$, i.e.,

$$M_m = S \ T \ R_x$$

then we need to multiply object A by the matrix

$$M = \begin{pmatrix} M_w & M_p & M_v \end{pmatrix} \begin{pmatrix} S & T & R_x \end{pmatrix}$$

$$= M_1 \begin{pmatrix} S & T & R_x \end{pmatrix}$$
But if object B only needs to be transformed by the translation $T$ and the scaling $S$, not the x-roll $R_x$, i.e.,

$$M_m = S \ T$$

then we need to multiply object B by the matrix

$$M = \begin{pmatrix} M_w & M_p & M_v \end{pmatrix} ( S \ T )$$

$$= M_1 ( S \ T )$$

The question is: how would you construct the new $M$ from the previous $M$?

(unless we keep a copy of $M_1$ and $M_m$ in addition to $M$)
Alternative:

Use the following $M$ to multiply object B first

$$M = (M_w M_p M_v) (S\ T)$$

$$= M_1 (S\ T)$$

Then use the following $M$ to transform object A

$$M \leftarrow M \ R_x$$

Question: can this approach solve all the problems?
Using Affine Transformations in a Program

2D:

Set up world window;
Set up viewport;
Initialize modelview matrix;

Define transformation #n;
Define transformation #(n-1);
...
Define transformation #1;
Send the graphics object;
Examples:

```c
...  
glMatrixMode ( GL_MODELVIEW );  
glLoadIdentity( );  //set CT to identity  
house ( );  
glMatrixMode ( GL_MODELVIEW );  
glRotated ( -30.0, 0.0, 0.0, 1.0 );  //rotation about z  
glMatrixMode ( GL_MODELVIEW );  
glTranslated ( 32.0, 25.0, 0.0 );  //translation in xy plane  
house ( );
```
5.4 Saving the CT for Later Use

Consider the following example:

If object A is transformed by a x-roll $R_x$, a translation $T$ and a scaling $S$, i.e.,

$$M_m = S \ T \ R_x$$

we need to multiply object A by the matrix

$$M = ( \begin{bmatrix} M_w & M_p & M_v \end{bmatrix} ) ( S \ T \ R_x )$$

$$= M_1 ( S \ T \ R_x )$$
Now, in addition to the x-roll $R_x$, the translation $T$ and the scaling $S$, if object B also needs to be transformed by a y-roll $R_y$, i.e.,

$$M_m = R_y \ S \ T \ R_x$$

then we need to multiply object B by the matrix

$$M = ( M_w \ M_p \ M_v ) ( R_y \ S \ T \ R_x )$$

$$= M_1 ( R_y \ S \ T \ R_x )$$

But how should the new $M$ be constructed?
Solution:

Push $M_3$;

Construct $M_4 (= M_3 S T R_x)$ for object A;

Remove (Pop) $M_4$;

Construct $M_5 (= M_3 R_y S T R_x)$ for object B;
Example: tiling based on a motif

```cpp
glPushMatrix();
glTranslated( W, H, 0 );
for (row = 0; row < 8; row++) {
    glPushMatrix();
    for (col = 0; col < 12; col++) {
        motif();
        glTranslated( L, 0.0, 0.0 ); // move to the right
    }
    glPopMatrix(); // back to the start
    glTranslated( 0.0, D, 0.0 ); // move up
}
glPopMatrix();
```
5.5 Drawing 3D Scenes with OpenGL

First, understand the OpenGL pipeline:

1. Position and aim the camera
2. Set the view volume shape
3. Define world window and viewport
Position and aim the camera

```c
glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

gluLookAt(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z);
```
What does `gluLookAt( )` do?

`n = eye − look`

`u = up × n`

`v = n × u`

gluLookAt then normalizes all three of these vectors, building the matrix

\[
M_v = \begin{bmatrix}
u_x & u_y & u_z & d_x \\
v_x & v_y & v_z & c_y \\
n_x & n_y & n_z & d_z \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

with \((d_x, d_y, d_z) = (−eye \cdot u, −eye \cdot v, −eye \cdot n)\) to convert world coordinates into eye coordinates.
Set the view volume shape

```c
glMatrixMode(GL_PROJECTION);

glLoadIdentity();

glOrtho(left, right, bottom, top, near, far);
```
How 3D transformations are used in an OpenGL-based program?

Example:

```
glPushMatrix();

glTranslated ( 0, len/2, 0 );

glScaled (thick, len, thick );

 glutSolidCube ( 1.0 );

 glPopMatrix();
```

Read and test sample programs: `wireframe.cpp` and `shade.cpp`